Machine-Checked Semantic Session Typing

Jonas Kastberg Hinrichsen, IT University of Copenhagen

Joint work with
Daniël Louwrink, University of Amsterdam
Robbert Krebbers, Radboud University
Jesper Bengtson, IT University of Copenhagen

January 18, 2021
CPP'21, Virtual, Denmark

Session Types – A Type System for Message Passing

Syntax

$$
\begin{array}{rll}
S ::= & \mathbf{!}A.\,S & | \\
 & \mathbf{?}A.\,S & | \\
 & \texttt{end} & | \ldots
\end{array}
$$

Session Types – A Type System for Message Passing

Syntax

$$
\begin{array}{rll}
S ::= & \mathbf{!}A.\,S & \mid \\
& \mathbf{?}A.\,S & \mid \\
& \mathtt{end} & \mid \ldots
\end{array}
$$

Type example

$$\mathbf{?}\mathsf{Z}.\,\mathbf{!}\mathsf{Z}.\,\mathtt{end}$$

Session Types – A Type System for Message Passing

Syntax

$$S ::= \mathbf{!}A.\,S \mid \mathbf{?}A.\,S \mid \texttt{end} \mid \ldots$$

Usage

$$c : \texttt{chan}\ S$$

Type example

$$\mathbf{?}\texttt{Z}.\,\mathbf{!}\texttt{Z}.\,\texttt{end}$$

Session Types – A Type System for Message Passing

Syntax

$$S ::= \mathbf{!}A.\,S \mid \mathbf{?}A.\,S \mid \texttt{end} \mid \ldots$$

Type example

$$\mathbf{?}\mathrm{Z}.\,\mathbf{!}\mathrm{Z}.\,\texttt{end}$$

Usage

$$c : \texttt{chan}\ S$$

Rule example

$$\Gamma, (x : \texttt{chan}\ (\mathbf{!}A.\,S)), (y : A) \vdash \texttt{send}\ x\ y : 1 \dashv \Gamma, (x : \texttt{chan}\ S)$$

Session Types – A Type System for Message Passing

Syntax

$$S ::= \mathbf{!}A.\,S \mid \mathbf{?}A.\,S \mid \texttt{end} \mid \ldots$$

Type example

$$\mathbf{?}\mathrm{Z}.\,\mathbf{!}\mathrm{Z}.\,\texttt{end}$$

Usage

$$c : \texttt{chan}\ S$$

Rule example

$$\Gamma, (x : \texttt{chan}\ (\mathbf{!}A.\,S)), (y : A) \vdash \texttt{send}\ x\ y : 1 \dashv \Gamma, (x : \texttt{chan}\ S)$$
$$\Gamma, (x : \texttt{chan}\ (\mathbf{?}A.\,S)) \vdash \texttt{recv}\ x : A \dashv \Gamma, (x : \texttt{chan}\ S)$$

Problems

1. **Lack of feature-rich session type systems**
   - Polymorphism, recursion, and subtyping have been studied individually
   - No session type systems that combines all three

Problems

1. **Lack of feature-rich session type systems**
   - Polymorphism, recursion, and subtyping have been studied individually
   - No session type systems that combines all three
2. **No support for "racy" yet safe programs**
   - Session type systems enforce a strict ownership discipline of channels
   - No way to type check safe use of exclusive resources

$$\lambda c.\,(\texttt{recv}\ c \mid\mid \texttt{recv}\ c) : \texttt{chan}\ (\texttt{?Z}.\,\texttt{?Z}.\,\texttt{end}) \multimap (\mathrm{Z} \times \mathrm{Z})$$

Problems

1. **Lack of feature-rich session type systems**
   - Polymorphism, recursion, and subtyping have been studied individually
   - No session type systems that combines all three
2. **No support for "racy" yet safe programs**
   - Session type systems enforce a strict ownership discipline of channels
   - No way to type check safe use of exclusive resources

     $\lambda\, c.\, (\texttt{recv}\ c \parallel \texttt{recv}\ c) : \texttt{chan}\ (\texttt{?Z}.\, \texttt{?Z}.\, \texttt{end}) \multimap (\mathrm{Z} \times \mathrm{Z})$
3. **Lack of mechanised soundness proofs for session type systems**
   - Few results exist for simpler systems
   - None exist for more expressive systems

Semantic typing

Semantic typing [Milner, Ahmed, Princeton PCC project, RustBelt project]

- Type system defined in terms of language semantics
- Modernly defined in terms of a program logic
- Expressivity and soundness inherited from underlying logic
- Allows manually proving safe yet untypeable programs

Semantic typing using **Iris**

Semantic typing [Milner, Ahmed, Princeton PCC project, RustBelt project]

- Type system defined in terms of language semantics
- Modernly defined in terms of a program logic
- Expressivity and soundness inherited from underlying logic
- Allows manually proving safe yet untypeable programs

Iris [Iris project]

- Higher-order concurrent separation logic
- Mechanised in **Coq**, with tactic support

Semantic typing using **Iris** and **Actris**

Semantic typing [Milner, Ahmed, Princeton PCC project, RustBelt project]

- Type system defined in terms of language semantics
- Modernly defined in terms of a program logic
- Expressivity and soundness inherited from underlying logic
- Allows manually proving safe yet untypeable programs

Iris [Iris project]

- Higher-order concurrent separation logic
- Mechanised in **Coq**, with tactic support

Actris [Hinrichsen et al. POPL'20]

- **Dependent separation protocols:** Logical protocols inspired by session types
- Mechanised in **Coq**, with tactic support

Semantic Typing

A **semantic type system** is defined in terms of the language semantics

Semantic Typing

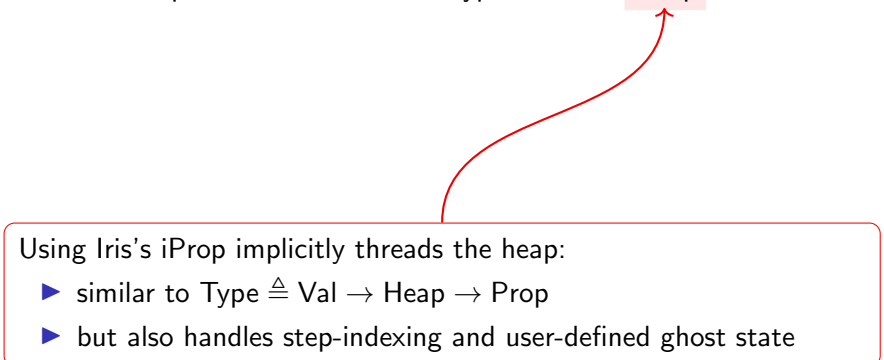
A **semantic type system** is defined in terms of the language semantics

- **Types** defined as predicates over values

Semantic Typing

A **semantic type system** is defined in terms of the language semantics

- **Types** defined as predicates over values $\text{Type} \triangleq \text{Val} \to \text{iProp}$



Semantic Typing

A **semantic type system** is defined in terms of the language semantics

- **Types** defined as predicates over values $\mathsf{Type} \triangleq \mathsf{Val} \to \mathsf{iProp}$

$$\mathsf{Z} \triangleq \lambda\, w.\, w \in \mathbb{Z}$$

Semantic Typing

A **semantic type system** is defined in terms of the language semantics

- **Types** defined as predicates over values $\text{Type} \triangleq \text{Val} \to \text{iProp}$

$$\mathsf{Z} \triangleq \lambda\, w.\, w \in \mathbb{Z}$$
$$A_1 \times A_2 \triangleq \lambda\, w.\, \exists w_1, w_2.\, (w = (w_1, w_2)) * (A_1\ w_1) * (A_2\ w_2)$$

Iris's *separation conjunction* $(P * Q)$ states that P and Q hold for disjoint parts of the heap

Semantic Typing

A **semantic type system** is defined in terms of the language semantics

- **Types** defined as predicates over values $\mathsf{Type} \triangleq \mathsf{Val} \to \mathsf{iProp}$

$$\mathsf{Z} \triangleq \lambda\, w.\, w \in \mathbb{Z}$$
$$A_1 \times A_2 \triangleq \lambda\, w.\, \exists w_1, w_2.\, (w = (w_1, w_2)) * (A_1\, w_1) * (A_2\, w_2)$$

- **Judgement** *defined* as safety-capturing evaluation

Semantic Typing

A **semantic type system** is defined in terms of the language semantics

- **Type** Iris's *weakest precondition* (wp $e\,\{\Phi\}$):
  - captures that safe e and $\forall v.\, e \longrightarrow^* v$ then $\Phi\ v$
  - safe e means "execution of e does not go wrong"
- **Judgement** *defined* as safety-capturing evaluation

  $\vDash e : A \triangleq \mathsf{wp}\ e\,\{A\}$

Semantic Typing

A **semantic type system** is defined in terms of the language semantics

- **Type**

Iris's *weakest precondition* (wp $e\,\{\Phi\}$):

- captures that safe e and $\forall v.\, e \longrightarrow^* v$ then $\Phi\ v$
- safe e means "execution of e does not go wrong"

- **Judgement** *defined* as safety-capturing evaluation

$$\vDash e : A \triangleq \text{wp}\, e\, \{A\}$$

- **Rules** are *proven* as lemmas:

Semantic Typing

A **semantic type system** is defined in terms of the language semantics

- **Type** Iris's *weakest precondition* (wp $e\,\{\Phi\}$):
  - captures that safe e and $\forall v.\, e \longrightarrow^* v$ then $\Phi\ v$
  - safe e means "execution of e does not go wrong"
- **Judgement** *defined* as safety-capturing evaluation

  $\vDash e : A \triangleq \text{wp}\, e\,\{A\}$
- **Rules** are *proven* as lemmas: $\vDash i : \mathsf{Z}$

Semantic Typing

A **semantic type system** is defined in terms of the language semantics

- **Type** Iris's *weakest precondition* (wp $e\,\{\Phi\}$):
  - captures that safe e and $\forall v.\, e \longrightarrow^* v$ then $\Phi\ v$
  - safe e means "execution of e does not go wrong"
- **Judgement** *defined* as safety-capturing evaluation

  $\vDash e : A \triangleq \text{wp}\, e\,\{A\}$
- **Rules** are *proven* as lemmas: $\vDash i : \mathsf{Z} \quad \rightsquigarrow \quad i \in \mathbb{Z}$

Semantic Typing

A **semantic type system** is defined in terms of the language semantics

- **Type**

Iris's *weakest precondition* $(\text{wp}\ e\ \{\Phi\})$:

- captures that safe e and $\forall v.\, e \longrightarrow^* v$ then $\Phi\ v$
- safe e means "execution of e does not go wrong"

- **Judgement** *defined* as safety-capturing evaluation

$$\vDash e : A \triangleq \text{wp}\ e\ \{A\}$$

- **Rules** are *proven* as lemmas: $\vDash i : \mathsf{Z} \quad \rightsquigarrow \quad i \in \mathbb{Z}$
- **Semantic type safety**

Semantic Typing

A **semantic type system** is defined in terms of the language semantics

- **Type** Iris's *weakest precondition* $(\text{wp}\ e\ \{\Phi\})$:
  - captures that safe e and $\forall v.\, e \longrightarrow^* v$ then $\Phi\ v$
  - safe e means "execution of e does not go wrong"
- **Judgement** *defined* as safety-capturing evaluation

  $\vDash e : A \triangleq \text{wp}\ e\ \{A\}$
- **Rules** are *proven* as lemmas: $\vDash i : \mathsf{Z} \rightsquigarrow i \in \mathbb{Z}$
- **Semantic type safety**: If $\vDash e : A$ then safe e

Semantic Typing

A **semantic type system** is defined in terms of the language semantics

- **Type** Iris's *weakest precondition* (wp $e\,\{\Phi\}$):
  - captures that safe e and $\forall v.\, e \longrightarrow^* v$ then $\Phi\ v$
  - safe e means "execution of e does not go wrong"
- **Judgement** *defined* as safety-capturing evaluation

  $\vDash e : A \triangleq \text{wp}\, e\, \{A\}$
- **Rules** are *proven* as lemmas: $\vDash i : \mathsf{Z} \quad \rightsquigarrow \quad i \in \mathbb{Z}$
- **Semantic type safety**: If $\vDash e : A$ then safe e
  - Consequence of the judgement definition

Semantic Typing

A **semantic type system** is defined in terms of the language semantics

- **Type** Iris's *weakest precondition* (wp $e\,\{\Phi\}$):
  - captures that safe e and $\forall v.\, e \longrightarrow^* v$ then $\Phi\ v$
  - safe e means "execution of e does not go wrong"
- **Judgement** *defined* as safety-capturing evaluation

  $\vDash e : A \triangleq \text{wp}\, e\,\{A\}$
- **Rules** are *proven* as lemmas: $\vDash i : \mathsf{Z} \rightsquigarrow i \in \mathbb{Z}$
- **Semantic type safety**: If $\vDash e : A$ then safe e
  - Consequence of the judgement definition

Semantic Session Types

Session types as a new type kind:

$$\begin{aligned} \mathsf{Type}_{\blacklozenge} &\triangleq ? \\ \mathbf{!}A.\,S &\triangleq ? \\ \mathbf{?}A.\,S &\triangleq ? \\ \mathtt{end} &\triangleq ? \end{aligned}$$

Semantic Session Types

Session types as a new type kind:

$$\begin{aligned} \mathsf{Type}_{\blacklozenge} &\triangleq ? \\ \mathbf{!}A.\, S &\triangleq ? \\ \mathbf{?}A.\, S &\triangleq ? \\ \mathtt{end} &\triangleq ? \end{aligned} \qquad \begin{aligned} \mathsf{Type}_{\bigstar} &\triangleq \mathsf{Val} \to \mathsf{iProp} \\ \mathtt{chan}\ S &\triangleq \lambda\, w.\, ? \end{aligned}$$

Actris Dependent Separation Protocols (Hinrichsen *et al.*, POPL'20)

Session type-inspired protocols for functional correctness

Actris Dependent Separation Protocols (Hinrichsen *et al.*, POPL'20)

Session type-inspired protocols for functional correctness, describing exchanges of:

- Logical variables

Actris Dependent Separation Protocols (Hinrichsen *et al.*, POPL'20)

Session type-inspired protocols for functional correctness, describing exchanges of:

- Logical variables
- Physical values

Actris Dependent Separation Protocols (Hinrichsen *et al.*, POPL'20)

Session type-inspired protocols for functional correctness, describing exchanges of:

- Logical variables
- Physical values
- Propositions / ownership

Actris Dependent Separation Protocols (Hinrichsen *et al.*, POPL'20)

Session type-inspired protocols for functional correctness, describing exchanges of:

- Logical variables
- Physical values
- Propositions / ownership

	Dependent separation protocols	**Session types**
Example	$\mathbf{?}\,(x : \mathbb{Z})\,\langle x \rangle \{\mathsf{True}\}.\,\mathbf{!}\,(y : \mathbb{Z})\,\langle y \rangle \{y = x + 2\}.\,\mathbf{end}$	$\mathbf{?}\mathsf{Z}.\,\mathbf{!}\mathsf{Z}.\,\mathtt{end}$
Usage	$c \rightarrowtail \mathit{prot}$	c : $\mathtt{chan}\ S$

Semantic Session Types

Session types as dependent separation protocols (iProto):

$$\begin{aligned} \mathrm{Type}_{\blacklozenge} &\triangleq \mathrm{iProto} \\ !A.\, S &\triangleq \mathbf{!}\,(v : \mathrm{Val})\,\langle v\rangle\{A\,v\}.\, S \\ ?A.\, S &\triangleq \mathbf{?}\,(v : \mathrm{Val})\,\langle v\rangle\{A\,v\}.\, S \\ \mathtt{end} &\triangleq \mathbf{end} \end{aligned} \qquad \begin{aligned} \mathrm{Type}_{\bigstar} &\triangleq \mathrm{Val} \to \mathrm{iProp} \\ \mathtt{chan}\ S &\triangleq \lambda\, w.\, w \rightarrowtail S \end{aligned}$$

Dependent separation protocols:

Example: $\mathbf{?}\,(x : \mathbb{Z})\,\langle x\rangle\{\mathsf{True}\}.\,\mathbf{!}\,(y : \mathbb{Z})\,\langle y\rangle\{y = x + 2\}.\,\mathbf{end}$

Usage: $c \rightarrowtail \mathit{prot}$

Semantic Session Types – Example Proof

Rule:

$$\Gamma, (x : \texttt{chan}\ (?A.\, S)) \vDash \texttt{recv}\ x : A \dashv \Gamma, (x : \texttt{chan}\ S)$$

Proof:

```
Lemma ltyped_recv Γ x A S :
  Γ !! x = Some (chan (<??> TY A; S))%lty →
  Γ ⊨ recv x : A ⊣ ctx_cons x (chan S) Γ.
Proof.
  iIntros (HΓx%ctx_lookup_perm) "!>".
  iIntros (σ) "HΓ /=". rewrite {1}HΓx /=.
  iDestruct (ctx_ltyped_cons with "HΓ") as
    (c Hσ) "[Hc HΓ]".
  rewrite Hσ.
  wp_recv (v) as "HA".
  iFrame "HA".
  iApply ctx_ltyped_cons; eauto with iFrame.
Qed.
```

Semantic Session Types – Example Proof

Rule:

$$\Gamma, (x : \texttt{chan}\ (?A.\, S)) \vDash \texttt{recv}\ x : A \dashv \Gamma, (x : \texttt{chan}\ S)$$

Proof:

```
Lemma ltyped_recv Γ x A S :
  Γ !! x = Some (chan (<??> TY A; S))%lty →
  Γ ⊨ recv x : A ⊣ ctx_cons x (chan S) Γ.
Proof.
  iIntros (HΓx%ctx_lookup_perm) "!>".
  iIntros (σ) "HΓ /=". rewrite {1}HΓx /=.
  iDestruct (ctx_ltyped_cons with "HΓ") as
    (c Hσ) "[Hc HΓ]".
  rewrite Hσ.
  wp_recv (v) as "HA".
  iFrame "HA".
  iApply ctx_ltyped_cons; eauto with iFrame.
Qed.
```

- Substructural resources are handled using **Iris**

Semantic Session Types – Example Proof

Rule:

$$\Gamma, (x : \texttt{chan}\ (?A.\, S)) \vDash \texttt{recv}\ x : A \dashv \Gamma, (x : \texttt{chan}\ S)$$

Proof:

```
Lemma ltyped_recv Γ x A S :
  Γ !! x = Some (chan (<??> TY A; S))%lty →
  Γ ⊨ recv x : A ⊣ ctx_cons x (chan S) Γ.
Proof.
  iIntros (HΓx%ctx_lookup_perm) "!>".
  iIntros (σ) "HΓ /=". rewrite {1}HΓx /=.
  iDestruct (ctx_ltyped_cons with "HΓ") as
    (c Hσ) "[Hc HΓ]".
  rewrite Hσ.
  wp_recv (v) as "HA".
  iFrame "HA".
  iApply ctx_ltyped_cons; eauto with iFrame.
Qed.
```

- Substructural resources are handled using **Iris**
- Message-passing is handled using **Actris**

Semantic Session Types – Example Proof

Rule:

$$\Gamma, (x : \texttt{chan}\ (?A.\, S)) \vDash \texttt{recv}\ x : A \dashv \Gamma, (x : \texttt{chan}\ S)$$

Proof:

```
Lemma ltyped_recv Γ x A S :
  Γ !! x = Some (chan (<??> TY A; S))%lty →
  Γ ⊨ recv x : A ⊣ ctx_cons x (chan S) Γ.
Proof.
  iIntros (HΓx%ctx_lookup_perm) "!>".
  iIntros (σ) "HΓ /=". rewrite {1}HΓx /=.
  iDestruct (ctx_ltyped_cons with "HΓ") as
    (c Hσ) "[Hc HΓ]".
  rewrite Hσ.
  wp_recv (v) as "HA".
  iFrame "HA".
  iApply ctx_ltyped_cons; eauto with iFrame.
Qed.
```

- Substructural resources are handled using **Iris**
- Message-passing is handled using **Actris**
- Mechanised in Coq using the **Iris Proof Mode**

Contributions

1. **Feature-rich session type systems**
   - We combined polymorphism, recursion, (asynchronous) subtyping, and more
   - By exploiting the expressivity of Iris and Actris

Contributions

1. **Feature-rich session type systems**
   - We combined polymorphism, recursion, (asynchronous) subtyping, and more
   - By exploiting the expressivity of Iris and Actris
2. **Support for "racy" yet safe programs**
   - We extend the type system with ad-hoc rules for "racy" programs like
     $$\Gamma \vDash \lambda\, c.\, (\texttt{recv}\ c \mid\mid \texttt{recv}\ c) : \texttt{chan}\ (\texttt{?Z}.\, \texttt{?Z}.\, \texttt{end}) \multimap (\mathsf{Z} \times \mathsf{Z}) \dashv \Gamma$$
   - By unfolding the definitions and using Iris ghost mechanisms

Contributions

1. **Feature-rich session type systems**
   - We combined polymorphism, recursion, (asynchronous) subtyping, and more
   - By exploiting the expressivity of Iris and Actris
2. **Support for "racy" yet safe programs**
   - We extend the type system with ad-hoc rules for "racy" programs like
     $$\Gamma \vDash \lambda\, c.\, (\texttt{recv}\ c \mid\mid \texttt{recv}\ c) : \texttt{chan}\ (?\mathrm{Z}.\, ?\mathrm{Z}.\, \texttt{end}) \multimap (\mathrm{Z} \times \mathrm{Z}) \dashv \Gamma$$
   - By unfolding the definitions and using Iris ghost mechanisms
3. **Mechanised soundness proof of our results**
   - We mechanised it in Coq: https://gitlab.mpi-sws.org/iris/actris/-/tree/cpp21
   - By building on top of Iris and Actris frameworks and libraries
   - Stable artifact: https://zenodo.org/record/4322752